GOOGLE SUMMER OF CODE - 2018

The FreeBSD Project

Audit Framework TestSuite Ideas

Here are the initial list of brainstormed ideas for developing Testsuite for the FreeBSD Audit Framework.

Fuzzing of OpenBSM audit viewer API

OpenBSM provides two audit viewer applications, `praudit`(1) & `auditreduce`(1) to print and select the records from the audit trail. These utilities would be tested against a set of fuzzing tools to ensure that corrupted/malicious audit-trail files can't compromise them. It is better to be robust against corrupted audit-trail files rather than having to give up on parsing them.

`Tool:` CERT's BFF `Difficulty:` Medium, test cases depend on tool's compatibility

Initial Smoke Testing

The Audit framework uses Sun's Basic Security Module (BSM) API for event auditing and configuring/creating trails (logs) and allowing users to control the audit daemon using tools like **audit(4) , auditreduce(1)** and **praudit(1).** We can test the working of these utilities to see if they generate correct usage messages (in case of wrong arguments) and work as expected in case of correct command line parameters.

Tool: Kyua(1) & atf-sh(3) Difficulty: Easy

Tool specific Kyua Testing

After Smoke testing, a particular set of tests can be developed to check the detailed functioning of all aforementioned APIs. This will be specific to each tool and will have to be developed manually.

Tool: Kyua(1) & atf-sh(3) Difficulty: Medium

Explicit System Call Testing (Suggested by Robert Watson)

The test application would trigger all Syscalls one by one, evaluating that the audit record contains all the expected parameters, e.g the arguments, valid argument types, return values etc. The testing will be done for various success and failure modes, with cross checking for appropriate error codes in case of failure mode.

Tool: C++ / Shell Difficulty: Medium, but would take considerable amount of time depending on number of syscalls tested.

Error Testing In Audit Configuration FIles

The audit configuration files present in `/etc/security` have some predefined rules for explicitly mentioning the events, rules, classes and many more parameters for smooth auditing process. However, an accidental misconfiguration by the administrator might lead to the collapse of the whole audit infrastructure. A testing can be done to ensure that these files are in sync with the defined rules.

Tool: Shell/Lua Scripting Difficulty: Easy, but might take some time

Event Selection Expression (Audit Class)

Link to Class Expressions

Selection expressions are used in a number of places in the audit configuration to determine which events should be audited. Expressions contain a list of event classes to match. A test automation can be done to ensure each of the **`20`** class expressions behave as expected and log the events they are supposed to audit.

Tool: Shell/Lua Scripting Difficulty: Rigorous and will take considerable amount of time

Audit-Control Parameter Evaluation

Link to audit-control parameters

The audit_control file `/etc/security/audit_control` has a list of system parameters which can be specified according to the need of administrator. E.g **dir, dist, flags, minfree** etc. This test will check if each parameter has feasible values. Parameters can be parsed from the man-page of audit_control(5).

Tool: Shell Scripting DIfficulty: Easy

Low Disk Space Warning Test for Minfree Parameter

The Audit framework has an option to specify the minimum space required for the audit logs to be generated, falling below which should issue a warning. `Minfree` parameter will be set to current remaining disk space and corresponding generation of warning will successfully pass this test.

Tool: Any Difficulty: Trivial

Read Access to Audit Trails

Only the members of group `audit` have the access to read the audit trails and manipulate it. This functionality can be checked by dropping the `setuid` bit of the current (root) user to a random user and trying to read the log trails. Error in doing so will result in success of this test.

Tool: C++/ Shell Difficulty: Moderate

Format of BSM/text/XML File Output (Modification suggestion by Robert Watson)

This test will check if the audit-reduction of log file has all the necessary information regarding the event audited like **header, event, attribute, subject, return, trailer** etc. The tests might vary depending on the attribution of the event, audit records and audit classes. Necessary semantic check will also be done.

Tool: Kyua(1) or Shell Difficulty: Moderate

Live monitoring of Audit Pipes

Audit pipes are cloning pseudo-devices which allow applications to tap the live audit record stream. This is primarily of interest to authors of intrusion detection and system monitoring applications. However, it might be the case that changes in configuration may not affect the way audits are live streamed. The testing for proper functioning of Audit piping on numerous scenarios can ascertain if the result is being generated properly.

Tool: Undecided DIfficulty: Might require some effort

NOTE: These are the initial test case scenarios I came up with. I might keep adding some more test cases and delete/edit the above tests according to need and on discussion with my mentors.

FINAL AUTOMATION TESTSUITE

As the tests are being developed, I will simultaneously work on integrating the individual scenarios in a Automated TestSuite Infrastructure. Resulting in a final tool which can be run by the administrator as and when required, covering all the aforementioned tests and producing a presentable output, with details on passed, failed tests and how to mitigate them.

IMPROVEMENTS IN FREEBSD AUDIT FRAMEWORK *

Linux has an Audit framework which works on the similar principle as FreeBSD. However, rather than the administrator having to manually edit the `auditd.conf/audit_control` file, there is a command line tool called `auditctl` which has plethora of options to configure the audit daemon. Apart from this, there are numerous tools in the Linux Audit Framework which ease the process of configuring/monitoring/presenting the event audit trails. Here is the brief description of each tool:-

- **auditd**: daemon to capture events and store them (log file)
- **auditctl**: client tool to configure auditd

- **audispd**: daemon to multiplex events
- **aureport**: reporting tool which reads from log file (auditd.log)
- **ausearch**: event viewer (auditd.log)
- **autrace**: using audit component in kernel to trace binaries
- **aulast**: similar to last, but instead using audit framework
- **aulastlog**: similar to lastlog, also using audit framework instead
- **ausyscall**: map syscall ID and name
- **auvirt**: displaying audit information regarding virtual machines

Introduction of such toolsuite in FreeBSD can significantly increase the efficiency and utility of the Audit Framework and would eliminate the need to test the 4th scenario as described in this report.

(* Can be considered a separate project in itself, but interesting. Will work on it post Audit testing.)